This notebook demonstrates the basic steps of text preprocessing.

- converting all letters to lower or upper case
- converting numbers into words or removing numbers
- removing punctuations, accent marks and other diacritics
- removing white spaces
- expanding abbreviations
- removing stop words, sparse terms, and particular words
- text canonicalization

Convert text to lowercase

simply use the standard python string (str) package

```
input_str = "The 5 biggest countries by population in this world are China, India, United States, Indonesia, and Brazil."
input_str = input_str.lower()
print(input_str)
```

```
the 5 biggest countries by population in this world are china, india, united states, indonesia, and brazil.
```

Remove numbers

use the regex (re) package to remove numbers

```
import re

input_str = " A sample space contains 3 red balls, 5 white balls, 7 pink balls, and 2 blue balls."
result = re.sub(r'\d+', '', input_str)
print(result)
```

```
 A sample space contains  red balls,  white balls,  pink balls, and blue balls.
```

Remove punctuation

use the string package to remove a predefined set of symbols [!"#$%&'()*+,-./:;<=>?@[]^_`{|}~]:

```
import string
input_str = "This &is [an] example? {of} string. with.?+-*=%& many ?@ punctuation!!!!"
# result = input_str.translate(str.maketrans(string.punctuation,""))
result = [c for c in input_str if c not in string.punctuation]
```

```
result = ''.join(result)
print(result)
```

```
This is an example of string with many  punctuation
```

Remove whitespaces

```
input_str = " \t a string example with white space "
print(f'Before whitespace removed: {input_str}')
input_str = input_str.strip()
print(f'After whitespace removed: {input_str}')
```

```
Before whitespace removed:        a string example with white space
After whitespace removed: a string example with white space
```

Remove stop words

Stop words are the most common words in a language like “the”, “a”, “on”, “is”, “all”. These words do not carry important meaning and are usually removed from texts. It is possible to remove stop words using Natural Language Toolkit (NLTK)

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words= stopwords.words('english')
input_str = "NLTK is a leading platform for building Python programs
to work with human language data."
tokens = word_tokenize(input_str)
print(f'Before removing: {tokens}')
result = [i for i in tokens if not i in stop_words]
print(f'After removing: {result}')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Before removing: ['NLTK', 'is', 'a', 'leading', 'platform', 'for',
'building', 'Python', 'programs', 'to', 'work', 'with', 'human',
'language', 'data', '.']
After removing: ['NLTK', 'leading', 'platform', 'building', 'Python',
'programs', 'work', 'human', 'language', 'data', '.']
```

Part of speech tagging (POS)

POS aims to assign parts of speech to each word of a given text (such as nouns, verbs, adjectives, and others) based on its definition and its context.

```
nltk.download('averaged_perceptron_tagger')

input_str = "Parts of speech examples: an article, to write,
interesting, easily, and, of"
tokens = word_tokenize(input_str)
pos_tags = nltk.pos_tag(tokens)
print(pos_tags)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('Parts', 'NNS'), ('of', 'IN'), ('speech', 'NN'), ('examples',
'NNS'), (':', ':'), ('an', 'DT'), ('article', 'NN'), (',', ','),
('to', 'TO'), ('write', 'VB'), (',', ','), ('interesting', 'VBG'),
(',', ','), ('easily', 'RB'), (',', ','), ('and', 'CC'), (',', ','),
('of', 'IN')]
```

Chunking (shallow parsing)

Chunking is a natural language process that identifies constituent parts of sentences (nouns, verbs, adjectives, etc.) and links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.)

```
# first, we add pos tags to the text - define the part of the speech
for each word
input_str = "A black television and a white stove were bought for the
new apartment of John."
tokens = word_tokenize(input_str)
pos_tags = nltk.pos_tag(tokens)

# second, chunking the text
reg_exp = "NP: {<DT>?<JJ>*<NN>}"
rp = nltk.RegexpParser(reg_exp)
result = rp.parse(pos_tags)
print(result)
```

```
(S
  (NP A/DT black/JJ television/NN)
  and/CC
  (NP a/DT white/JJ stove/NN)
  were/VBD
  bought/VBN
  for/IN
  (NP the/DT new/JJ apartment/NN)
```

```
  of/IN
  John/NNP
  ./.)
```

Named entity recognition

Named-entity recognition (NER) aims to find named entities in text and classify them into pre-defined categories (names of persons, locations, organizations, times, etc.).

```
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download('maxent_ne_chunker')
nltk.download('words')

input_str = "Bill works for Apple so he went to Boston for a
conference."
print(ne_chunk(pos_tag(word_tokenize(input_str))))
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
(S
  (PERSON Bill/NNP)
  works/VBZ
  for/IN
  Apple/NNP
  so/IN
  he/PRP
  went/VBD
  to/TO
  (GPE Boston/NNP)
  for/IN
  a/DT
  conference/NN
  ./.)
```